A Nearly Linear-Time PTAS for Explicit Fractional Packing and Covering Linear Programs

Christos Koufogiannakis · Neal E. Young

Abstract We give an approximation algorithm for fractional packing and covering linear programs (linear programs with non-negative coefficients). Given a constraint matrix with n non-zeros, r rows, and c columns, the algorithm (with high probability) computes feasible primal and dual solutions whose costs are within a factor of $1+\varepsilon$ of OPT (the optimal cost) in time $O((r+c)\log(n)/\varepsilon^2+n)$.[1]

1 Introduction

A *packing* problem is a linear program of the form $\max\{a \cdot x : Mx \leq b, x \in P\}$, where the entries of the constraint matrix M are non-negative and P is a convex polytope admitting some form of optimization oracle. A *covering* problem is of the form $\min\{a \cdot \hat{x} : M\hat{x} \geq b, \hat{x} \in P\}$.

This paper focuses on *explicitly given* packing and covering problems, that is, $\max\{a \cdot x : Mx \leq b, x \geq 0\}$ and $\min\{a \cdot \hat{x} : M\hat{x} \geq b, \hat{x} \geq 0\}$, where the polytope P is just the positive orthant. Explicitly given packing and covering are important special cases of linear programming, including, for example, fractional set cover, multicommodity flow problems with given paths, two-player zero-sum matrix games with non-negative payoffs, and variants of these problems.

The paper gives a $(1+\varepsilon)$-approximation algorithm — that is, an algorithm that returns feasible primal and dual solutions whose costs are within a given factor $1+\varepsilon$ of OPT. With high probability, it runs in time $O((r+c)\log(n)/\varepsilon^2+n)$, where n – the input size – is the number of non-zero entries in the constraint matrix and $r+c$ is the number of rows plus columns (i.e., constraints plus variables).

For dense instances, $r+c$ can be as small as $O(\sqrt{n})$. For moderately dense instances – as long as $r+c = o(n/\log n)$ – the $1/\varepsilon^2$ factor multiplies a sub-linear term. Generally, the time is linear in the input size n as long as $\varepsilon \geq \Omega(\sqrt{(r+c)\log(n)/n})$.

1.1 Related work

The algorithm is a Lagrangian-relaxation (a.k.a. price-directed decomposition, multiplicative weights) algorithm. Broadly, these algorithms work by replacing a set of hard constraints by a sum of smooth penalties, one per constraint, and then iteratively augmenting a solution while trading off the increase in the objective against the increase in the sum of penalties. Here the penalties are exponential in the constraint violation, and, in each iteration, only the first-order (linear) approximation is used to estimate the change in the sum of penalties.

Such algorithms, which can provide useful alternatives to interior-point and Simplex methods, have a long history and a large literature. Bienstock gives an implementation-oriented, operations-research perspective [2]. Arora *et al.* discuss them from a computer-science perspective, highlighting connections to other fields such as learning theory [1]. An overview by Todd places them in the context of general linear programming [18].

Department of Computer Science and Engineering, University of California, Riverside. The first author would like to thank the Greek State Scholarship Foundation (IKY). The second author's research was partially supported by NSF grants 0626912, 0729071, and 1117954.

[1] Accepted to Algorithmica, 2013. The conference version of this paper was "Beating Simplex for fractional packing and covering linear programs" [13].

The running times of algorithms of this type increase as the approximation parameter ε gets small. For algorithms that rely on linear approximation of the penalty changes in each iteration, the running times grow at least quadratically in $1/\varepsilon$ (times a polynomial in the other parameters). For explicitly given packing and covering, the fastest previous such algorithm that we know of runs in time $O((r+c)\bar{c}\log(n)/\varepsilon^2)$, where $\bar{c}$ is the maximum number of columns in which any variable appears [21]. That algorithm applies to *mixed* packing and covering — a more general problem. Using some of the techniques in this paper, one can improve that algorithm to run in time $O(n\log(n)/\varepsilon^2)$ (an unpublished result), which is slower than the algorithm here for dense problems.

Technically, the starting point for the work here is a remarkable algorithm by Grigoriadis and Khachiyan for the following special case of packing and covering [9]. The input is a two-player zero-sum matrix game with payoffs in $[-1,1]$. The output is a pair of mixed strategies that guarantee an expected payoff within an *additive* ε of optimal. (Note that achieving additive error ε is, however, easier than achieving multiplicative error $1+\varepsilon$.) The algorithm computes the desired output in $O((r+c)\log(n)/\varepsilon^2)$ time. This is remarkable in that, for dense matrices, it is *sub-linear* in the input size $n=\Theta(rc)$.[2] (For a machine-learning algorithm closely related to Grigoriadis and Khachiyan's result, see [5,6].)

We also use the idea of *non-uniform increments* from algorithms by Garg and Könemann [8,12,7].

Dependence on ε. Building on work by Nesterov (e.g., [16,17]), recent algorithms for packing and covering problems have reduced the dependence on $1/\varepsilon$ from quadratic to linear, at the expense of increased dependence on other parameters. Roughly, these algorithms better approximate the change in the penalty function in each iteration, leading to fewer iterations but more time per iteration (although not to the same extent as interior-point algorithms). For example, Bienstock and Iyengar give an algorithm for concurrent multicommodity flow that solves $O^*(\varepsilon^{-1}k^{1.5}|V|^{0.5})$ shortest-path problems, where k is the number of commodities and $|V|$ is the number of vertices [3]. Chudak and Eleuterio continue this direction — for example, they give an algorithm for fractional set cover running in worst-case time $O^*(c^{1.5}(r+c)/\varepsilon + c^2 r)$ [4].

Comparison to Simplex and Interior-Point methods. Currently, the most commonly used algorithms for solving linear programs in practice are Simplex and interior-point methods. Regarding Simplex algorithms, commercial implementations algorithms use many carefully tuned heuristics (e.g. pre-solvers and heuristics for maintaining sparsity and numerical stability), enabling them to quickly solve many practical problems with millions of non-zeros to optimality. But, as is well known, their worst-case running times are exponential. Also, for both Simplex and interior-point methods, running times can vary widely depending on the structure of the underlying problem. (A detailed analysis of Simplex and interior-point running times is outside the scope of this paper.) These issues make rigorous comparison between the various algorithms difficult.

Still, here is a meta-argument that may allow some meaningful comparison. Focus on "square" constraint matrices, where $r=\Theta(c)$. Note that at a minimum, any Simplex implementation must identify a non-trivial basic feasible solution. Likewise, interior-point algorithms require (in each iteration) a Cholesky decomposition or other matrix factorization. Thus, essentially, both methods require implicitly (at least) solving an $r\times r$ system of linear equations. Solving such a system is a relatively well-understood problem, both in theory and in practice, and (barring special structure) takes $\Omega(r^3)$ time, or $\Omega(r^{2.8})$ time using Strassen's algorithm. Thus, on "square" instances, Simplex and interior-point algorithms should have running times growing at least with $\Omega(r^{2.8})$ (and probably more). This reasoning applies even if Simplex or interior-point methods are terminated early so as to find *approximately* optimal solutions.

In comparison, on "square" matrices, the algorithm in this paper takes time $O(n+r\log(r)/\varepsilon^2)$ where $n=O(r^2)$ or less. If the meta-argument holds, then, for applications where $(1+\varepsilon)$-approximate solutions suffice for some fixed and moderate ε (say, $\varepsilon\approx 1\%$), for very large instances (say, $r\geq 10^4$), the algorithm here should be orders of magnitude faster than Simplex or interior-point algorithms.

This conclusion is consistent with experiments reported here, in which the running times of Simplex and interior-point algorithms on large random instances exceed $\Omega(r^{2.8})$. Concretely, with $\varepsilon=1\%$, the algorithm here is faster when r is on the order of 10^3, with a super-linear (in r) speed-up for larger r.

[2] The problem studied here, packing and covering, can be reduced to Grigoriadis and Khachiyan's problem. This reduction leads to an $O((r+c)\log(n)(U\ \mathrm{OPT})^2/\varepsilon^2)$-time algorithm to find a $(1+\varepsilon)$-approximate packing/covering solution, where $U \doteq \max_{ij} M_{ij}/(b_i a_j)$. A pre-processing step [14, §2.1] can bound U, leading to a running time bound of $O((r+c)\log(n)\min(r,c)^4/\varepsilon^4)$.

1.2 Technical roadmap

Broadly, the running times of iterative optimization algorithms are determined by (1) the number of iterations and (2) the time per iteration. Various algorithms trade off these two factors in different ways. The technical approach taken here is to accept a high number of iterations — $(r+c)\log(n)/\varepsilon^2$, a typical bound for an algorithm of this class (see e.g. [11] for further discussion) — and to focus on implementing each iteration as quickly as possible (ideally in constant amortized time).

Coupling. Grigoriadis and Khachiyan's sub-linear time algorithm uses an unusual technique of *coupling* primal and dual algorithms that is critical to the algorithm here. As a starting point, to explain coupling, consider the following "slow" coupling algorithm. (Throughout, assume without loss of generality by scaling that $a_j = b_i = 1$ for all i, j.) The algorithm starts with all-zero primal and dual solutions, x and $\hat{x}$, respectively. In each iteration, it increases one coordinate x_j of the primal solution x by 1, and increases one coordinate $\hat{x}_i$ of the dual solution $\hat{x}$ by 1. The index j of the primal variable to increment is chosen randomly from a distribution $\hat{p}$ that depends on the current *dual* solution. Likewise, the index i of the dual variable to increment is chosen randomly from a distribution p that depends on the current *primal* solution. The distribution $\hat{p}$ is concentrated on the indices of dual constraints $M^\intercal\hat{x}$ that are "most violated" by $\hat{x}$. Likewise, the distribution p is concentrated on the indices of primal constraints Mx that are "most violated" by x. Specifically, p_i is proportional to $(1+\varepsilon)^{M_i x}$, while $\hat{p}_j$ is proportional to $(1-\varepsilon)^{M_j^\intercal \hat{x}}$.[3]

Lemma 1 in the next section proves that this algorithm achieves the desired approximation guarantee. Here, broadly, is why coupling helps reduce the time per iteration in comparison to the standard approach. The standard approach is to increment the primal variable corresponding to a dual constraint that is "most violated" by p — that is, to increment $x_{j'}$ where j' (approximately) minimizes $M_{j'}^\intercal p$ (for p defined as above). This requires at a minimum *maintaining* the vector $M^\intercal p$. Recall that p_i is a function of $M_i x$. Thus, a change in one primal variable $x_{j'}$ changes many entries in the vector p, but *even more* entries in $M^\intercal p$. (In the $r \times c$ bipartite graph $G = ([r],[c],E)$ where $E = \{(i,j) : M_{ij} \neq 0\}$, the neighbors of j' change in p, while all *neighbors of those neighbors* change in $M^\intercal p$.) Thus, maintaining $M^\intercal p$ is costly. In comparison, to implement coupling, it is enough to maintain the vectors p and $\hat{p}$. The further product $M^\intercal p$ is not needed (nor is $M\hat{p}$). This is the basic reason why coupling helps reduce the time per iteration.

Non-uniform increments. The next main technique, used to make more progress per iteration, is Garg and Könemann's *non-uniform increments* [8,12,7]. Instead of incrementing the primal and dual variables by a uniform amount each time (as described above), the algorithm increments the chosen primal and dual variables $x_{j'}$ and $\hat{x}_{i'}$ by an amount $\delta_{i'j'}$ chosen small enough so that the left-hand side (LHS) of each constraint (each $M_i x$ or $M_j^\intercal \hat{x}$) increases by at most 1 (so that the analysis still holds), but *large enough* so that the LHS of *at least one* such constraint increases by at least $1/4$. This is small enough to allow the same correctness proof to go through, but is large enough to guarantee a small number of iterations. The number of iterations is bounded by (roughly) the following argument: each iteration increases the LHS of some constraint by $1/4$, but, during the course of the algorithm, no LHS ever exceeds $N \approx \log(n)/\varepsilon^2$. (The particular N is chosen with foresight so that the relative error works out to $1+\varepsilon$.) Thus, the number of iterations is $O((r+c)N) = O((r+c)\log(n)/\varepsilon^2)$.

Using slowly changing estimates of Mx and $M^\intercal\hat{x}$. In fact, we will achieve this bound not just for the number of iterations, but also for the total work done (outside of pre- and post-processing). The key to this is the third main technique. Most of the work done by the algorithm as described so far would be in maintaining the vectors Mx and $M^\intercal\hat{x}$ and the distributions p and $\hat{p}$ (which are functions of Mx and $M^\intercal\hat{x}$). This would require lots of time in the worst case, because, even with non-uniform increments, there can still be many *small* changes in elements of Mx and $M^\intercal\hat{x}$. To work around this, instead of maintaining Mx and $M^\intercal\hat{x}$ exactly, the algorithm maintains more slowly changing *estimates* for them (vectors y and $\hat{y}$, respectively), using random sampling. The algorithm maintains $y \approx Mx$ as follows. When the algorithm increases a primal variable $x_{j'}$ during an iteration, this increases some elements in the vector Mx (specifically, the elements $M_i x$ where $M_{ij} > 0$). For each such element $M_i x$, if the element increases by, say, $\delta \leq 1$, then the algorithm increases the corresponding y_i not by δ, but by 1, but *only with probability* δ. This maintains not only $E[y_i] = M_i x$, but also, with high probability, $y_i \approx M_i x$. Further, the algorithm only does work for a y_i (e.g. updating p_i) when y_i increases (by 1). The algorithm maintains the estimate vector $\hat{y} \approx M^\intercal\hat{x}$ similarly, and defines the sampling distributions p and $\hat{p}$ as functions of y and $\hat{y}$ instead of Mx and $M^\intercal\hat{x}$. In this way

[3] The algorithm can be interpreted as a form of fictitious play of a two-player zero-sum game, where in each round each player plays from a distribution concentrated around the best response to the aggregate of the opponent's historical plays. In contrast, in many other fictitious-play algorithms, one or both of the player plays a *deterministic* pure best-response to the opponent's historical average.

each unit of work done by the algorithm can be charged to an increase in $|Mx| + |M^\intercal \hat{x}|$ (or more precisely, an increase in $|y| + |\hat{y}|$, which never exceeds $(r + c)N$). (Throughout the paper, $|v|$ denotes the 1-norm of any vector v.)

Section 2 gives the formal intuition underlying coupling by describing and formally analyzing the first (simpler, slower) coupling algorithm described above. Section 3 describes the full (main) algorithm and its correctness proof. Section 4 gives remaining implementation details and bounds the run time. Section 5 presents basic experimental results, including a comparison with the GLPK Simplex algorithm.

1.3 Preliminaries

For the rest of the paper, assume the primal and dual problems are of the following restricted forms, respectively: $\max\{|x| : Mx \le \mathbf{1}, x \ge 0\}$, $\min\{|\hat{x}| : M^\intercal \hat{x} \ge \mathbf{1}, \hat{x} \ge 0\}$. That is, assume $a_j = b_i = 1$ for each i, j. This is without loss of generality by the transformation $M'_{ij} = M_{ij}/(b_i a_j)$. Recall that $|v|$ denotes the 1-norm of any vector v.

2 Slow algorithm (coupling)

To illustrate the coupling technique, in this section we analyze the first (simpler but slower) algorithm described in the roadmap in the introduction, a variant of Grigoriadis and Khachiyan's algorithm [9]. We show that it returns a $(1 - 2\varepsilon)$-approximate primal-dual pair with high probability.

We do not analyze the running time, which can be large. In the following section, we describe how to modify this algorithm (using non-uniform increments and the random sampling trick described in the previous roadmap) to obtain the full algorithm with a good time bound.

For just this section, assume that each $M_{ij} \in [0, 1]$. (Assume as always that $b_i = a_j = 1$ for all i, j; recall that $|v|$ denotes the 1-norm of v.) Here is the algorithm:

slow-alg$(M \in [0,1]^{r \times c}, \varepsilon)$

1. Vectors $x, \hat{x} \leftarrow \mathbf{0}$; scalar $N = \lceil 2 \ln(rc)/\varepsilon^2 \rceil$.
2. Repeat until $\max_i M_i x \ge N$:
3. Let $p_i \doteq (1 + \varepsilon)^{M_i x}$ (for all i) and $\hat{p}_j \doteq (1 - \varepsilon)^{M_j^\intercal \hat{x}}$ (for all j).
4. Choose random indices j' and i' respectively from probability distributions $\hat{p}/|\hat{p}|$ and $p/|p|$.
5. Increase $x_{j'}$ and $\hat{x}_{i'}$ each by 1.
6. Let $(x^\star, \hat{x}^\star) \doteq (x/\max_i M_i x, \hat{x}/\min_j M_j^\intercal \hat{x})$.
7. Return $(x^\star, \hat{x}^\star)$.

The scaling of x and $\hat{x}$ in line 6 ensures feasibility of the final primal solution $x^\star$ and the final dual solution $\hat{x}^\star$. (Recall the assumption that $b_i = a_j = 1$ for all i, j.) The final primal solution cost and final dual solution costs are, respectively $|x^\star| = |x|/\max_i M_i x$ and $|\hat{x}^\star| = |\hat{x}|/\min_j M_j^\intercal \hat{x}$. Since the algorithm keeps the 1-norms $|x|$ and $|\hat{x}|$ of the intermediate primal and dual solutions equal, the final primal and dual costs will be within a factor of $1 - 2\varepsilon$ of each other as long as $\min_j M_j^\intercal \hat{x} \ge (1 - 2\varepsilon) \max_i M_i x$. If this event happens, then by weak duality implies that each solution is a $(1 - 2\varepsilon)$-approximation of its respective optimum.

To prove that the event $\min_j M_j^\intercal \hat{x} \ge (1 - 2\varepsilon) \max_i M_i x$ happens with high probability, we show that $|p| \cdot |\hat{p}|$ (the product of the 1-norms of p and $\hat{p}$, as defined in the algorithm) is a Lyapunov function — that is, the product is non-increasing in expectation with each iteration. Thus, its expected final value is at most its initial value rc, and with high probability, its final value is at most, say, $(rc)^2$. If that happens, then by careful inspection of p and $\hat{p}$, it must be that $(1 - \varepsilon) \max_i M_i x \le \min_j M_j^\intercal \hat{x} + \varepsilon N$, which (with the termination condition $\max_i M_i x \ge N$) implies the desired event.[4]

[4] It may be instructive to compare this algorithm to the more standard algorithm. In fact there are two standard algorithms related to this one: a primal algorithm and a dual algorithm. In each iteration, the primal algorithm would choose j' to minimize $M_{j'}^\intercal p$ and increments $x_{j'}$. Separately and simultaneously, the dual algorithm would choose i' to maximize $(M\hat{p})_{i'}$, then increments $\hat{x}_{i'}$. (Note that the primal algorithm and the dual algorithm are independent, and in fact either can be run without the other.) To prove the approximation ratio for the primal algorithm, one would bound the increase in $|p|$ relative to the increase in the primal objective $|x|$. To prove the approximation ratio for the dual algorithm,

Lemma 1 *The slow algorithm returns a $(1-2\varepsilon)$-approximate primal-dual pair (feasible primal and dual solutions $x^\star$ and $\hat{x}^\star$ such that $|x^\star| \geq (1-2\varepsilon)|\hat{x}^\star|$) with probability at least $1 - 1/(rc)$.*

Proof In a given iteration, let p and $\hat{p}$ denote the vectors at the start of the iteration. Let p' and $\hat{p}'$ denote the vectors at the end of the iteration. Let Δx denote the vector whose jth entry is the increase in x_j during the iteration (or if z is a scalar, Δz denotes the increase in z). Then, using that each $\Delta M_i x = M_{ij'} \in [0,1]$,

$$|p'| \;=\; \sum_i p_i(1+\varepsilon)^{M_i \Delta x} \;\leq\; \sum_i p_i(1+\varepsilon M_i \Delta x) \;=\; |p|\Big[1+\varepsilon \frac{p^\intercal}{|p|} M \Delta x\Big].$$

Likewise, for the dual, $|\hat{p}'| \leq |\hat{p}|[1-\varepsilon(\hat{p}/|\hat{p}|)^\intercal M^\intercal \Delta \hat{x}]$.

Multiplying these bounds on $|p'|$ and $|\hat{p}'|$ and using that $(1+a)(1-b) = 1+a-b-ab \leq 1+a-b$ for $a, b \geq 0$ gives

$$|p'||\hat{p}'| \;\leq\; |p||\hat{p}|\Big[\; 1 \;+\; \varepsilon \frac{p}{|p|}^\intercal M \Delta x \;-\; \varepsilon \Delta \hat{x}^\intercal M \frac{\hat{p}}{|\hat{p}|} \;\Big].$$

The inequality above is what motivates the "coupling" of primal and dual increments. The algorithm chooses the random increments to x and $\hat{x}$ precisely so that $\mathrm{E}[\Delta x] = \hat{p}/|\hat{p}|$ and $\mathrm{E}[\Delta \hat{x}] = p/|p|$. Taking expectations of both sides of the inequality above, and plugging these equations into the two terms on the right-hand side, the two terms exactly cancel, giving $\mathrm{E}[|p'||\hat{p}'|] \leq |p||\hat{p}|$. Thus, the particular random choice of increments to x and $\hat{x}$ makes the quantity $|p|\,|\hat{p}|$ non-increasing in expectation with each iteration.

This and Wald's equation (Lemma 9, or equivalently a standard optional stopping theorem for supermartingales) imply that the expectation of $|p||\hat{p}|$ at termination is at most its initial value rc. So, by the Markov bound, the probability that $|p||\hat{p}| \geq (rc)^2$ is at most $1/rc$. Thus, with probability at least $1 - 1/rc$, at termination $|p||\hat{p}| \leq (rc)^2$.

Assume this happens. Note that $(rc)^2 \leq \exp(\varepsilon^2 N)$, so $|p||\hat{p}| \leq (rc)^2$ implies $(1+\varepsilon)^{\max_i M_i x}(1-\varepsilon)^{\min_j M_j^\intercal \hat{x}} \leq |p||\hat{p}| \leq \exp(\varepsilon^2 N)$. Taking logs, and using the inequalities $1/\ln(1/(1-\varepsilon)) \leq 1/\varepsilon$ and $\ln(1+\varepsilon)/\ln(1/(1-\varepsilon)) \geq 1-\varepsilon$, gives $(1-\varepsilon)\max_i M_i x \leq \min_j M_j^\intercal \hat{x} + \varepsilon N$.

By the termination condition $\max_i M_i x \geq N$, so the above inequality implies $(1-2\varepsilon)\max_i M_i x \leq \min_j M_j^\intercal \hat{x}$.

This and $|x| = |\hat{x}|$ (and weak duality) imply the approximation guarantee for the primal-dual pair $(x^\star, \hat{x}^\star)$ returned by the algorithm. □

3 Full algorithm

This section describes the full algorithm and gives a proof of its approximation guarantee. In addition to the coupling idea explained in the previous section, for speed the full algorithm uses non-uniform increments and estimates of Mx and $M^\intercal \hat{x}$ as described in the introduction. Next we describe some more details of those techniques. After that we give the algorithm in detail (although some implementation details that are not crucial to the approximation guarantee are delayed to the next section).

Recall that WLOG we are assuming $a_i = b_j = 1$ for all i, j. The only assumption on M is $M_{ij} \geq 0$.

Non-uniform increments. In each iteration, instead of increasing the randomly chosen $x_{j'}$ and $\hat{x}_{i'}$ by 1, the algorithm increases them both by an increment $\delta_{i'j'}$, chosen just so that the maximum resulting increase in any left-hand side (LHS) of any constraint (i.e. $\max_i \Delta M_i x$ or $\max_j \Delta M_j^\intercal \hat{x}$) is in $[1/4, 1]$. The algorithm also deletes covering constraints once they become satisfied (the set J contains indices of not-yet-satisfied covering constraints, that is j such that $M_j^\intercal \hat{x} < N$).

We want the analysis of the approximation ratio to continue to hold (the analogue of Lemma 1 for the slow algorithm), even with the increments adjusted as above. That analysis requires that the expected change in each x_j and each $\hat{x}_i$ should be proportional to $\hat{p}_j$ and p_i, respectively. Thus, we adjust the sampling distribution for the random pair i', j' so that, when we choose i' and j' from the distribution and increment $x_{j'}$ and $\hat{x}_{i'}$ by $\delta_{i'j'}$ as defined above, it is the case that, for any i and j, $\mathrm{E}[\Delta x_j] = \alpha \hat{p}_j/|\hat{p}|$ and $\mathrm{E}[\Delta \hat{x}_i] = \alpha p_i/|p|$ for an $\alpha > 0$. This is done by scaling the probability of choosing each given i', j' pair by a factor proportional to $1/\delta_{i'j'}$.

---

one would bound the decrease in $|\hat{p}|$ relative to the increase in the dual objective $|\hat{x}|$. In this view, the coupled algorithm can be obtained by taking these two independent primal and dual algorithms and randomly coupling their choices of i' and j'. The analysis of the coupled algorithm uses as a penalty function $|p||\hat{p}|$, the product of the respective penalty functions $|p|, |\hat{p}|$ of the two underlying algorithms.

To implement the above non-uniform increments and the adjusted sampling distribution, the algorithm maintains the following data structures as a function of the current primal and dual solutions x and $\hat{x}$: a set J of indices of still-active (not yet met) covering constraints (columns); for each column $M_j^\intercal$ its maximum entry $u_j = \max_i M_{ij}$; and for each row M_i a close upper bound $\hat{u}_i$ on its maximum active entry $\max_{j \in J} M_{ij}$ (specifically, the algorithm maintains $\hat{u}_i \in [1,2] \times \max_{j \in J} M_{ij}$).

Then, the algorithm takes the increment $\delta_{i'j'}$ to be $1/(\hat{u}_{i'} + u_{j'})$. This seemingly odd choice has two key properties: (1) It satisfies $\delta_{i'j'} = \Theta(1/\max(\hat{u}_{i'}, u_{j'}))$, which ensures that when $x_{j'}$ and $\hat{x}_{i'}$ are increased by $\delta_{i'j'}$, the maximum increase in any LHS (any $M_i x$, or $M_j^\intercal \hat{x}$ with $j \in J$) is $\Omega(1)$. (2) It allows the algorithm to select the random pair (i', j') in constant time using the following subroutine, called random-pair (the notation $p \times \hat{u}$ denotes the vector with ith entry $p_i \hat{u}_i$):

random-pair$(p, \hat{p}, p \times \hat{u}, \hat{p} \times u)$

1. With probability $|p \times \hat{u}||\hat{p}|/(|p \times \hat{u}||\hat{p}| + |p||\hat{p} \times u|)$
   choose random i' from distribution $p \times \hat{u}/|p \times \hat{u}|$,
   and independently choose j' from $\hat{p}/|\hat{p}|$,
2. or, otherwise,
   choose random i' from distribution $p/|p|$,
   and independently choose j' from $\hat{p} \times u/|\hat{p} \times u|$.
3. Return (i', j').

The key property of **random-pair** is that it makes the expected changes in x and $\hat{x}$ correct: any given pair (i, j) is chosen with probability proportional to $p_i \hat{p}_j / \delta_{ij}$, which makes the expected change in any x_j and $\hat{x}_i$, respectively, is proportional to $\hat{p}_j$ and p_i. (See Lemma 2 below.)

Maintaining estimates (y and $\hat{y}$) of Mx and $M^\intercal \hat{x}$. Instead of maintaining the vectors p and $\hat{p}$ as direct functions of the vectors Mx and $M^\intercal \hat{x}$, to save work, the algorithm maintains more slowly changing *estimates* (y and $\hat{y}$) of the vectors Mx and $M^\intercal \hat{x}$, and maintains p and $\hat{p}$ as functions of the estimates, rather than as functions of Mx and $M^\intercal \hat{x}$.

Specifically, the algorithm maintains y and $\hat{y}$ as follows. When any $M_i x$ increases by some $\delta \in [0,1]$ in an iteration, the algorithm increases the corresponding estimate y_i by 1 with probability δ. Likewise, when any $M_j^\intercal \hat{x}$ increases by some $\hat{\delta} \in [0,1]$ in an iteration, the algorithm increases the corresponding estimate $\hat{y}_j$ by 1 with probability $\hat{\delta}$. Then, each p_i is maintained as $p_i = (1+\varepsilon)^{y_i}$ instead of $(1+\varepsilon)^{M_i x}$, and each $\hat{p}_j$ is maintained as $\hat{p}_j = (1-\varepsilon)^{\hat{y}_j}$ instead of $(1+\varepsilon)^{M_i x}$. This reduces the frequency of updates to p and $\hat{p}$ (and so reduces the total work), yet maintains $y \approx Mx$ and $\hat{y} \approx M^\intercal \hat{x}$ with high probability, which is enough to still allow a (suitably modified) coupling argument to go through.

Each change to a y_i or a $\hat{y}_j$ increases the changed element by 1. Also, no element of y or $\hat{y}$ gets larger than N before the algorithm stops (or the corresponding covering constraint is deleted). Thus, in total the elements of y and $\hat{y}$ are changed at most $O((r+c)N) = O((r+c)\log(n)/\varepsilon^2)$ times. We implement the algorithm to do only *constant work* maintaining the remaining vectors for each such change. This allows us to bound the total time by $O((r+c)\log(n)/\varepsilon^2)$ (plus $O(n)$ pre- and post-processing time).

As a step towards this goal, in each iteration, in order to *determine* the elements in y and $\hat{y}$ that change, using just $O(1)$ work per changed element, the algorithm uses the following trick. It chooses a random $\beta \in [0,1]$. It then increments y_i by 1 for those i such that the increase $M_{ij'}\delta_{i'j'}$ in $M_i x$ is at least β. Likewise, it increments $\hat{y}_j$ by 1 for j such that the increase $M_{i'j}\delta_{i'j'}$ in $M_j^\intercal \hat{x}$ is at least β. To do this efficiently, the algorithm preprocesses M, so that within each row M_i or column $M_j^\intercal$ of M, the elements can be accessed in (approximately) decreasing order in constant time per element accessed. (This preprocessing is described in Section 4.) This method of incrementing the elements of y and $\hat{y}$ uses constant work per changed element and increments each element with the correct probability. (The random increments of different elements are not independent, but this is okay because, in the end, each estimate y_j and $\hat{y}_i$ will be shown seperately to be correct with high probability.)

The detailed algorithm is shown in Fig. 1, except for the subroutine random-pair (above) and some implementation details that are left until Section 4.

Approximation guarantee. Next we state and prove the approximation guarantee for the full algorithm in Fig. 1. We first prove three utility lemmas. The first utility lemma establishes that (in expectation) x, $\hat{x}$, y, and $\hat{y}$ change as desired in each iteration.

solve$(M \in \mathbf{R}_+^{r\times c}, \varepsilon)$ — *return a* $(1-6\varepsilon)$*-approximate primal-dual pair w/ high prob.*

1. Initialize vectors $x, \hat{x}, y, \hat{y} \leftarrow \mathbf{0}$, and scalar $N = \lceil 2\ln(rc)/\varepsilon^2 \rceil$.
2. Precompute $u_j \doteq \max\{M_{ij} : i \in [r]\}$ for $j \in [c]$. (The max. entry in column M_j.)
   As x and $\hat{x}$ are incremented, the alg. maintains y and $\hat{y}$ so $\mathrm{E}[y] = Mx$, $\mathrm{E}[\hat{y}] = M^\intercal \hat{x}$.
   It maintains vectors p defined by $p_i \doteq (1+\varepsilon)^{y_i}$ and, as a function of $\hat{y}$:

$$J \doteq \{j \in [c] : \hat{y}_j \le N\} \qquad \text{(the active columns)}$$
$$\hat{u}_i \in [1,2] \times \max\{M_{ij} : j \in J\} \quad \text{(approximates the max. active entry in row } i \text{ of } M)$$
$$\hat{p}_j \doteq \begin{cases} (1-\varepsilon)^{\hat{y}_j} & \text{if } j \in J \\ 0 & \text{otherwise.} \end{cases}$$

   It maintains vectors $p \times \hat{u}$ and $\hat{p} \times u$, where $a \times b$ is a vector whose ith entry is $a_i b_i$.
3. Repeat until $\max_i y_i = N$ or $\min_j \hat{y}_j = N$:
4. Let $(i', j') \leftarrow$ **random-pair**$(p, \hat{p}, p \times \hat{u}, \hat{p} \times u)$.
5. Increase $x_{j'}$ and $\hat{x}_{i'}$ each by the same amount $\delta_{i'j'} \doteq 1/(\hat{u}_{i'} + u_{j'})$.
6. Update y, $\hat{y}$, and the other vectors as follows:
7. Choose random $\beta \in [0,1]$ uniformly, and
8. for each $i \in [r]$ with $M_{ij'}\delta_{i'j'} \ge \beta$, increase y_i by 1
9. (and multiply p_i and $(p \times \hat{u})_i$ by $1+\varepsilon$);
10. for each $j \in J$ with $M_{i'j}\delta_{i'j'} \ge \beta$, increase $\hat{y}_j$ by 1
11. (and multiply $\hat{p}_j$ and $(\hat{p} \times u)_j$ by $1-\varepsilon$).
12. For each j leaving J, update J, $\hat{u}$, and $p \times \hat{u}$.
13. Let $(x^\star, \hat{x}^\star) \doteq (x/\max_i M_i x, \hat{x}/\min_j M_j^\intercal \hat{x})$. Return $(x^\star, \hat{x}^\star)$.

Fig. 1 The full algorithm. $[i]$ denotes $\{1, 2, \ldots, i\}$. Implementation details are in Section 4.

Lemma 2 *In each iteration,*

1. *The largest change in any relevant LHS is at least 1/4:*

$$\max\{\max_i \Delta M_i x, \max_{j\in J} \Delta M_j^\intercal \hat{x}\} \;\in\; [1/4, 1].$$

2. *Let* $\alpha \doteq |p||\hat{p}|/\sum_{ij} p_i \hat{p}_j/\delta_{ij}$. *The expected changes in each* x_j, x_j, y_i, $\hat{y}_j$ *satisfy*

$$E[\Delta x_j] = \alpha \hat{p}_j/|\hat{p}|,\ \mathrm{E}[\Delta y_i] = \mathrm{E}[\Delta M_i x] = \alpha M \hat{p}_i/|\hat{p}|,$$
$$\mathrm{E}[\Delta \hat{x}_i] = \alpha p_i/|p|,\ \mathrm{E}[\Delta \hat{y}_j] = \mathrm{E}[\Delta M_j^\intercal \hat{x}] = \alpha M^\intercal p_j/|p|.$$

Proof (i) By the choice of $\hat{u}$ and u, for the (i', j') chosen, the largest change in a relevant LHS is

$$\begin{aligned} \delta_{i'j'} \max\big(\max_i M_{ij'}, \max_{j\in J} M_{i'j}\big) &\in [1/2, 1]\, \delta_{i'j'} \max(\hat{u}_{i'}, u_{j'}) \\ &\subseteq [1/4, 1]\, \delta_{i'j'}(\hat{u}_{i'} + u_{j'}) \\ &= [1/4, 1]. \end{aligned}$$

(ii) First, we verify that the probability that random-pair returns a given (i, j) is $\alpha(p_i/|p|)(\hat{p}_j/|\hat{p}|)/\delta_{ij}$. Here is the calculation. By inspection of random-pair, the probability is proportional to

$$|p \times \hat{u}|\,|\hat{p}| \frac{p_i \hat{u}_i}{|p \times \hat{u}|} \frac{\hat{p}_j}{|\hat{p}|} + |p|\,|\hat{p} \times u| \frac{p_i}{p} \frac{\hat{p}_j u_j}{|\hat{p} \times u|}$$

which by algebra simplifies to $p_i \hat{p}_j(\hat{u}_i + u_j) = p_i \hat{p}_j/\delta_{ij}$.

Hence, the probability must be $\alpha(p_i/|p|)(\hat{p}_j/|\hat{p}|)/\delta_{ij}$, because the choice of α makes the sum over all i and j of the probabilities equal 1.

Next, note that part (i) of the lemma implies that in line 8 (given the chosen i' and j') the probability that a given y_i is incremented is $M_{ij'}\delta_{i'j'}$, while in line 10 the probability that a given $\hat{y}_j$ is incremented is $M_{i'j}\delta_{i'j'}$.

Now, the remaining equalities in (ii) follow by direct calculation. For example:

$$\mathrm{E}[\Delta x_j] = \sum_i (\alpha p_i/|p|)(\hat{p}_j/|\hat{p}|)/\delta_{ij})\delta_{ij} = \alpha \hat{p}_j/|\hat{p}|. \qquad \square$$

The next lemma shows that (with high probability) the estimate vectors y and $\hat{y}$ suitably approximate Mx and $M^{\mathsf{T}}\hat{x}$, respectively. The proof is simply an application of an appropriate Azuma-like inequality (tailored to deal with the random stopping time of the algorithm).

Lemma 3

1. *For any i, with probability at least $1 - 1/(rc)^2$, at termination $(1-\varepsilon)M_i x \le y_i + \varepsilon N$.*
2. *For any j, with probability at least $1 - 1/(rc)^2$, after the last iteration with $j \in J$, it holds that $(1-\varepsilon)\hat{y}_j \le M_j^{\mathsf{T}}\hat{x} + \varepsilon N$.*

Proof (i) By Lemma 2, in each iteration each $M_i x$ and y_i increase by at most 1 and the expected increases in these two quantities are the same. So, by the Azuma inequality for random stopping times (Lemma 10), $\Pr[(1-\varepsilon)M_i x \ge y_i + \varepsilon N]$ is at most $\exp(-\varepsilon^2 N) \le 1/(rc)^2$. This proves (i).

The proof for (ii) is similar, noting that, while $j \in J$, the quantity $M_j^{\mathsf{T}}\hat{x}$ increases by at most 1 each iteration. □

Finally, here is the main utility lemma. Recall that the heart of the analysis of the slow algorithm (Lemma 1) was showing that in expectation $|p||\hat{p}|$ was non-increasing. This allowed us to conclude that (with high probability at the end) $\max_i M_i x$ was not much larger than $\min_j M_j^{\mathsf{T}}\hat{x}$. This was the key to proving the approximation ratio.

The next lemma gives the analogous argument for the full algorithm. It shows that the quantity $|p||\hat{p}|$ is non-increasing in expectation, which, by definition of p and $\hat{p}$, implies that (with high probability at the end) $\max_i y_i$ is not much larger than $\min_j \hat{y}_j$. The proof is essentially the same as that of Lemma 1, but with some technical complications accounting for the deletion of covering constraints.

Since (with high probability by Lemma 3) the estimates y and $\hat{y}$ approximate Mx and $M\hat{x}$, respectively, this implies that (with high probability at the end) $\max_i M_i x$ is not much larger than $\min_j M_j^{\mathsf{T}}\hat{x}$. Since the algorithm maintains $|x| = |\hat{x}|$, this is enough to prove the approximation ratio.

Lemma 4 *With probability at least $1 - 1/rc$, when the algorithm stops, $\max_i y_i \le N$ and $\min_j \hat{y}_j \ge (1-2\varepsilon)N$.*

Proof Let p' and $\hat{p}'$ denote p and $\hat{p}$ after a given iteration, while p and $\hat{p}$ denote the values before the iteration. We claim that, given p and $\hat{p}$, $\mathrm{E}[|p'|\,|\hat{p}'|] \le |p|\,|\hat{p}|$ — with each iteration $|p|\,|\hat{p}|$ is non-increasing in expectation. To prove it, note $|p'| = \sum_i p_i(1+\varepsilon\Delta y_i) = |p| + \varepsilon p^{\mathsf{T}}\Delta y$ and, similarly, $|\hat{p}'| = |\hat{p}| - \varepsilon\hat{p}^{\mathsf{T}}\Delta\hat{y}$ (recall $\Delta y_i, \Delta\hat{y}_j \in \{0,1\}$). Multiplying these two equations and dropping a negative term gives

$$|p'|\,|\hat{p}'| \;\le\; |p|\,|\hat{p}| + \varepsilon|\hat{p}|p^{\mathsf{T}}\Delta y - \varepsilon|p|\hat{p}^{\mathsf{T}}\Delta\hat{y}.$$

The claim follows by taking expectations of both sides, then, in the right-hand side applying linearity of expectation and substituting $\mathrm{E}[\Delta y] = \alpha M\hat{p}/|\hat{p}|$ and $\mathrm{E}[\Delta\hat{y}] = \alpha M^{\mathsf{T}}p/|p|$ from Lemma 2.

By Wald's equation (Lemma 9), the claim implies that $\mathrm{E}[|p|\,|\hat{p}|]$ for p and $\hat{p}$ at termination is at most its initial value rc. Applying the Markov bound, with probability at least $1 - 1/rc$, at termination $\max_i p_i \max_j \hat{p}_j \le |p||\hat{p}| \le (rc)^2 \le \exp(\varepsilon^2 N)$.

Assume this event happens. The index set J is not empty at termination, so the minimum $\hat{y}_j$ is achieved for $j \in J$. Substitute in the definitions of p_i and $\hat{p}_j$ and take log to get $\max_i y_i \ln(1+\varepsilon) \le \min_j \hat{y}_j \ln(1/(1-\varepsilon)) + \varepsilon^2 N$.

Divide by $\ln(1/(1-\varepsilon))$, apply $1/\ln(1/(1-\varepsilon)) \le 1/\varepsilon$ and also $\ln(1+\varepsilon)/\ln(1/(1-\varepsilon)) \ge 1-\varepsilon$. This gives $(1-\varepsilon)\max_i y_i \le \min_j \hat{y}_j + \varepsilon N$.

By the termination condition $\max_i y_i \le N$ is guaranteed, and either $\max_i y_i = N$ or $\min_j \hat{y}_j = N$. If $\min_j \hat{y}_j = N$, then the event in the lemma occurs. If not, then $\max_i y_i = N$, which (with the inequality in previous paragraph) implies $(1-\varepsilon)N \le \min_j \hat{y}_j + \varepsilon N$, again implying the event in the lemma. □

Finally, here is the approximation guarantee (Theorem 1). It follows from the three lemmas above by straightforward algebra.

Theorem 1 *With probability at least $1 - 3/rc$, the algorithm in Fig. 1 returns feasible primal and dual solutions $(x^\star, \hat{x}^\star)$ with $|x^\star|/|\hat{x}^\star| \ge 1 - 6\varepsilon$.*

Proof Recall that the algorithm returns $(x^\star, \hat{x}^\star) \doteq (x/\max_i M_i x, \hat{x}/\min_j M_j^{\mathsf{T}}\hat{x})$. By the naive union bound, with probability at least $1 - 3/rc$ (for all i and j) the events in Lemma 3 occur, and the event in Lemma 4 occurs. Assume all of these events happen. Then, at termination, for all i and j,

$$\begin{aligned}(1-\varepsilon)M_i x &\le y_i + \varepsilon N \\ y_i &\le N\end{aligned} \qquad \text{and} \qquad \begin{aligned}(1-2\varepsilon)N &\le \hat{y}_j \\ (1-\varepsilon)\hat{y}_j &\le M_j^\intercal \hat{x} + \varepsilon N.\end{aligned}$$

By algebra, using $(1-a)(1-b) \ge 1-a-b$ and $1/(1+\varepsilon) \ge 1-\varepsilon$, it follows for all i and j that

$$(1-2\varepsilon)M_i x \le N \qquad \text{and} \qquad (1-4\varepsilon)N \le M_j^\intercal \hat{x}.$$

This implies $\min_j M_j^\intercal \hat{x} / \max_i M_i x \ge 1-6\varepsilon$.

The scaling at the end of the algorithm assures that $x^\star$ and $\hat{x}^\star$ are feasible. Since the sizes $|x|$ and $|\hat{x}|$ increase by the same amount each iteration, they are equal. Thus, the ratio of the primal and dual objectives is $|x^\star|/|\hat{x}^\star| = \min_j M_j^\intercal \hat{x} / \max_i M_i x \ge 1-6\varepsilon$. □

4 Implementation details and running time

This section gives remaining implementation details for the algorithm and bounds the running time. The remaining implementation details concern the maintenance of the vectors $(x, \hat{x}, y, \hat{y}, p, \hat{p}, u, \hat{u}, p \times \hat{u}, \hat{p} \times u)$ so that each update to these vectors can be implemented in constant time and random-pair can be implemented in constant time.

The matrix M should be given in any standard sparse representation, so that the non-zero entries can be traversed in time proportional to the number of non-zero entries.

4.1 Simpler implementation

First, here is an implementation that takes $O(n \log n + (r+c)\log(n)/\varepsilon^2)$ time. (After this we describe how to modify this implementation to remove the $\log n$ factor from the first term.)

Theorem 2 *The algorithm can be implemented to return a $(1-6\varepsilon)$-approximate primal-dual pair for packing and covering in time $O(n \log n + (r+c)\log(n)/\varepsilon^2)$ with probability at least $1 - 4/rc$.*

Proof To support random-pair, store each of the four vectors $p, \hat{p}, p \times \hat{u}, \hat{p} \times u$ in its own random-sampling data structure [15] (see also [10]). This data structure maintains a vector v; it supports random sampling from the distribution $v/|v|$ and changing any entry of v in constant time. Then random-pair runs in constant time, and each update of an entry of p, $\hat{p}$, $p \times \hat{u}$, or $\hat{p} \times u$ takes constant time.

Updating the estimates y and $\hat{y}$ in each iteration requires, given i' and j', identifying which j and i are such that $M_{i'j}$ and $M_{ij'}$ are at least $\beta/\delta_{i'j'}$ (the corresponding elements y_i and $\hat{y}_j$ get increased). To support this efficiently, at the start of the algorithm, preprocess the matrix M. Build, for each row and column, a doubly linked list of the non-zero entries. *Sort each list in descending order.* Cross-reference the lists so that, given an entry M_{ij} in the ith row list, the corresponding entry M_{ij} in the jth column list can be found in constant time. The total time for preprocessing is $O(n \log n)$.

Now implement each iteration as follows. Let $\mathcal{I}_t$ denote the set of indices i for which y_i is incremented in line 8 in iteration t. From the random $\beta \in [0,1]$ and the sorted list for row j', compute this set $\mathcal{I}_t$ by traversing the list for row j' in order of decreasing $M_{ij'}$, collecting elements until an i with $M_{ij'} < \beta/\delta_{i'j'}$ is encountered. Then, for each $i \in \mathcal{I}_t$, update y_i, p_i, and the ith entry in $p \times \hat{u}$ in constant time. Likewise, let $\mathcal{J}_t$ denote the set of indices j for which $\hat{y}_j$ is incremented in line 10. Compute $\mathcal{J}_t$ from the sorted list for column i'. For each $j \in \mathcal{J}_t$, update $\hat{p}_j$, and the jth entry in $\hat{p} \times u$. The total time for these operations during the course of the algorithm is $O(\sum_t 1 + |\mathcal{I}_t| + |\mathcal{J}_t|)$.

For each element j that leaves J during the iteration, update $\hat{p}_j$. Delete all entries in the jth column list from all row lists. For each row list i whose first (largest) entry is deleted, update the corresponding $\hat{u}_i$ by setting $\hat{u}_i$ to be the next (now first and maximum) entry remaining in the row list; also update $(p \times \hat{u})_i$. The total time for this during the course of the algorithm is $O(n)$, because each M_{ij} is deleted at most once.

This completes the implementation.

By inspection, the total time is $O(n \log n)$ (for preprocessing, and deletion of covering constraints) plus $O(\sum_t 1 + |\mathcal{I}_t| + |\mathcal{J}_t|)$ (for the work done as a result of the increments).

The first term $O(n \log n)$ above is in its final form. The next three lemmas bound the second term (the sum). The first lemma bounds the sum except for the "1". That is, it bounds the number of times any y_i or $\hat{y}_j$ is incremented. (There are $r+c$ elements, and each can be incremented at most N times during the course of the algorithm.)

Lemma 5

$$\sum_t |\mathcal{I}_t| + |\mathcal{J}_t| \le (r+c)N \;=\; O((r+c)\log(n)/\varepsilon^2).$$

Proof First, $\sum_t |\mathcal{I}_t| \le rN$ because each y_i can be increased at most N times before $\max_i y_i \ge N$ (causing termination). Second, $\sum_t |\mathcal{J}_t| \le cN$ because each $\hat{y}_j$ can be increased at most N times before j leaves J and ceases to be updated. □

The next lemma bounds the remaining part of the second term, which is $O(\sum_t 1)$. Given that $\sum_t |\mathcal{I}_t| + |\mathcal{J}_t| \le (r+c)N$, it's enough to bound the number of iterations t where $|\mathcal{I}_t| + |\mathcal{J}_t| = 0$. Call such an iteration *empty*. (The 1's in the non-empty iterations contribute at most $\sum_t |\mathcal{I}_t| + |\mathcal{J}_t| \le (r+c)N$ to the sum.)

We first show that each iteration is non-empty with probability at least 1/4. This is so because, for any (i', j') pair chosen in an iteration, for the constraint that determines the increment $\delta_{i'j'}$, the expected increase in the corresponding y_i or $\hat{y}_j$ must be at least 1/4, and that element will be incremented (making the iteration non-empty) with probability at least 1/4.

Lemma 6 *Given the state at the start of an iteration, the probability that it is empty is at most 3/4.*

Proof Given the (i', j') chosen in the iteration, by (1) of Lemma 2, by definition of $\delta_{i'j'}$, there is either an i such that $M_{ij'}\delta_{i'j'} \ge 1/4$ or a j such that $M_{i'j}\delta_{i'j'} \ge 1/4$. In the former case, $i \in \mathcal{I}_t$ with probability at least 1/4. In the latter case, $j \in \mathcal{J}_t$ with probability at least 1/4. □

This implies that, with high probability, the number of empty iterations does not exceed three times the number of non-empty iterations by much. (This follows from the Azuma-like inequality.) We have already bounded the number of non-empty iterations, so this implies a bound (with high probability) on the number of empty iterations.

Lemma 7 *With probability at least $1 - 1/rc$, the number of empty iterations is $O((r+c)N)$.*

Proof Let E_t be 1 for empty iterations and 0 otherwise. By the previous lemma and the Azuma-like inequality tailored for random stopping times (Lemma 10), for any $\delta, A \ge 0$,

$$\Pr\Big[\ (1-\delta)\textstyle\sum_{t=1}^{T} E_t \;\ge\; 3\sum_{t=1}^{T}(1-E_t) \;+\; A\ \Big] \;\le\; \exp(-\delta A).$$

Taking $\delta = 1/2$ and $A = 2\ln(rc)$, it follows that with probability at least $1 - 1/rc$, the number of empty iterations is bounded by a constant times the number of non-empty iterations plus $2\ln(rc)$. The number of non-empty iterations is at most $(r+c)N$, hence, with probability at least $1 - 1/rc$ the number of empty iterations is $O((r+c)N)$. □

Finally we complete the proof of Theorem 2, stated at the top of the section.

As discussed above, the total time is $O(n \log n)$ (for preprocessing, and deletion of covering constraints) plus $O(\sum_t 1 + |\mathcal{I}_t| + |\mathcal{J}_t|)$ (for the work done as a result of the increments).

By Lemma 5, $\sum_t |\mathcal{I}_t| + |\mathcal{J}_t| = O((r+c)\log(n)/\varepsilon^2)$. By Lemma 7, with probability $1 - 1/rc$, the number of iterations t such that $|\mathcal{I}_t| + |\mathcal{J}_t| = 0$ is $O((r+c)\log(n)/\varepsilon^2)$. Together, these imply that, with probability $1 - 1/rc$, and the total time is $O(n \log n + (r+c)\log(n)/\varepsilon^2)$. This and Theorem 1 imply Theorem 2. □

4.2 Faster implementation.

To prove the main result, it remains to describe how to remove the $\log n$ factor from the $n \log n$ term in the time bound in the previous section.

The idea is that it suffices to *approximately* sort the row and column lists, and that this can be done in linear time.

Theorem 3 *The algorithm can be implemented to return a $(1 - 7\varepsilon)$-approximate primal-dual pair for packing and covering in time $O(n + (c+r)\log(n)/\varepsilon^2)$ with probability at least $1 - 5/rc$.*

Proof Modify the algorithm as follows.

First, preprocess M as described in [14, §2.1] so that the non-zero entries have bounded range. Specifically, let $\beta = \min_j \max_i M_{ij}$. Let $M'_{ij} \doteq 0$ if $M_{ij} < \beta\varepsilon/c$ and $M'_{ij} \doteq \min\{\beta c/\varepsilon, M_{ij}\}$ otherwise. As shown in [14], any $(1-6\varepsilon)$-approximate primal-dual pair for the transformed problem will be a $(1-7\varepsilon)$-approximate primal-dual pair for the original problem.

In the preprocessing step, instead of sorting the row and column lists, *pseudo-sort* them — sort them based on keys $\lfloor \log_2 M_{ij} \rfloor$. These keys will be integers in the range $\log_2(\beta) \pm \log(c/\varepsilon)$. Use bucket sort, so that a row or column with k entries can be processed in $O(k + \log(c/\varepsilon))$ time. The total time for pseudo-sorting the rows and columns is $O(n + (r+c)\log(c/\varepsilon))$.

Then, in the tth iteration, maintain the data structures as before, except as follows.

Compute the set $\mathcal{I}_t$ as follows. Traverse the pseudo-sorted jth column until an index i with $M_{ij'}\delta_{i'j'} < \beta/2$ is found. (No indices later in the list can be in $\mathcal{I}_t$.) Take all the indices i seen with $M_{ij'}\delta_{i'j'} \geq \beta$. Compute the set $\mathcal{J}_t$ similarly. Total time for this is $O(\sum_t 1 + |\mathcal{I}'_t| + |\mathcal{J}'_t|)$, where $\mathcal{I}'_t$ and $\mathcal{J}'_t$ denote the sets of indices actually traversed (so $\mathcal{I}_t \subseteq \mathcal{I}'_t$ and $\mathcal{J}_t \subseteq \mathcal{J}'_t$).

When an index j leaves the set J, delete all entries in the jth column list from all row lists. For each row list affected, set $\hat{u}_i$ to *two* times the first element remaining in the row list. This ensures $\hat{u}_i \in [1,2] \max_{j \in J} M_{ij}$.

These are the only details that are changed.

The total time is now $O(n + (r+c)\log(c/\varepsilon))$ for preprocessing and deletion of covering constraints, plus $O(\sum_t 1 + |\mathcal{I}'_t| + |\mathcal{J}'_t|)$ to implement the increments and vector updates. To finish, the next lemma bounds the latter term. The basic idea is that, in each iteration, each matrix entry is at most *twice* as likely to be examined as it was in the previous algorithm. Thus, with high probability, each matrix element is examined at most about twice as often as it would have been in the previous algorithm.

Lemma 8 *With probability at least* $1 - 2/rc$*, it happens that* $\sum_t (1 + |\mathcal{I}'_t| + |\mathcal{J}'_t|) = O((r+c)N)$.

Proof Consider a given iteration. Fix i' and j' chosen in the iteration. For each i, note that, for the random $\beta \in [0,1]$,

$$\begin{aligned} \Pr[i \in \mathcal{I}'_t] \;\leq\; \Pr[\beta/2 \leq M_{ij'}\delta_{i'j'}] \;&\leq\; 2M_{ij'}\delta_{i'j'} \\ &=\; 2\Pr[\beta \leq M_{ij'}\delta_{i'j'}] \;=\; 2\Pr[i \in \mathcal{I}_t]. \end{aligned}$$

Fix an i. Applying Azuma-like inequality for random stopping times (Lemma 10), for any $\delta, A \geq 0$,

$$\Pr\Big[\; (1-\delta)\textstyle\sum_t [i \in \mathcal{I}'_t] \;\geq\; 2\sum_t [i \in \mathcal{I}_t] \;+\; A \;\Big] \;\leq\; \exp(-\delta A).$$

(Above $[i \in S]$ denotes 1 if $i \in S$ and 0 otherwise.)

Taking $\delta = 1/2$ and $A = 4\ln(rc)$, with probability at least $1 - (rc)^2$, it happens that

$$\textstyle\sum_t [i \in \mathcal{I}'_t] \;\leq\; 4\sum_t [i \in \mathcal{I}_t] \;+\; 8\ln(rc).$$

Likewise, for any j, with probability at least $1 - 1/(rc)^2$, we have that $\sum_t [j \in \mathcal{J}'_t] \;\leq\; 2\sum_t [j \in \mathcal{J}_t] \;+\; 8\ln(rc)$.

Summing the naive union bound over all i and j, with probability at least $1 - 1/rc$, it happens that the sum $\sum_t (|\mathcal{I}'_t| + |\mathcal{J}'_t|)$ is at most $4\sum_t (|\mathcal{I}_t| + |\mathcal{J}_t|) \;+\; 8(r+c)\ln(rc)$.

By Lemma 5 the latter quantity is $O((r+c)N)$.

By Lemma 7, the number of empty iterations is still $O((r+c)N)$ with probability at least $1 - 1/rc$. The lemma follows by applying the naive union bound. □

If the event in the lemma happens, then the total time is $O(n + (r+c)\log(n)/\varepsilon^2)$. This proves Theorem 3. □

5 Empirical Results

We performed an experimental evaluation of our algorithm and compared it against Simplex on randomly generated 0/1 input matrices. These experiments suffer from the following limitations: (i) the instances are relatively small, (ii) the instances are random and thus not representative of practical applications, (iii) the comparison is to the publicly available GLPK (GNU Linear Programming Kit), not the industry standard CPLEX. With those caveats, here are the findings.

The running time of our algorithm is well-predicted by the analysis, with a leading constant factor of about 12 basic operations in the big-O term in which ε occurs.

For moderately large inputs, the algorithm can be substantially faster than Simplex (GLPK – Gnu Linear Programming Kit – Simplex algorithm glpsol version 4.15 with default options).[5] *The empirical running times reported here for Simplex are to find a* $(1 \pm \varepsilon)$*-approximate solution.*

For inputs with 2500-5000 rows and columns, the algorithm (with $\varepsilon = 0.01$) is faster than Simplex by factors ranging from tens to hundreds. For larger instances, the speedup grows roughly linearly in rc. For instances with moderately small ε and thousands (or more) rows and columns, the algorithm is orders of magnitude faster than Simplex.

The test inputs had $r, c \in [739, 5000]$, $\varepsilon \in \{0.02, 0.01, 0.005\}$, and matrix density $d \in \{1/2^k : k = 1, 2, 3, 4, 5, 6\}$. For each (r, c, d) tuple there was a random 0/1 matrix with r rows and c columns, where each entry was 1 with probability d. The algorithm here was run on each such input, with each ε. The running time was compared to that taken by a Simplex solver to find a $(1 - \varepsilon)$-approximate solution.

GLPK Simplex failed to finish due to cycling on about 10% of the initial runs; those inputs are excluded from the final data. This left 167 runs. The complete data for the non-excluded runs is given in the tables at the end of the section.

5.1 Empirical evaluation of this algorithm

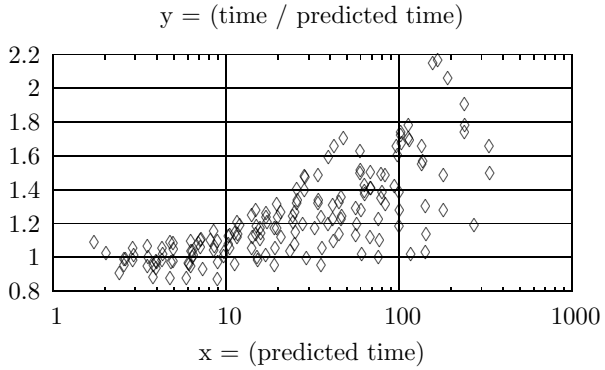
The running time of the algorithm here includes (A) time for preprocessing and initialization, (B) time for sampling (line 4, once per iteration of the outer loop), and (C) time for increments (lines 8 and 10, once per iteration of the inner loops). Theoretically the dominant terms are $O(n)$ for (A) and $O((r+c)\log(n)/\varepsilon^2)$ for (C). For the inputs tested here, the significant terms in practice are for (B) and (C), with the role of (B) diminishing for larger instances. The time (number of basic operations) is well-predicted by the expression

$$[12(r+c) \ + \ 480d^{-1}]\frac{\ln(rc)}{\varepsilon^2} \tag{1}$$

where $d = 1/2^k$ is the density (fraction of matrix entries that are non-zero, at least $1/\min(r, c)$).

The $12(r+c)\ln(rc)/\varepsilon^2$ term is the time spent in (C), the inner loops; it is the most significant term in the experiments as r and c grow. The less significant term $480d^{-1}\ln(rc)/\varepsilon^2$ is for (B), and is proportional to the number of samples (that is, iterations of the outer loop). Note that this term *decreases* as matrix density increases. (For the implementation we focused on reducing the time for (C), not for (B). It is probable that the constant 480 above can be reduced with a more careful implementation.)

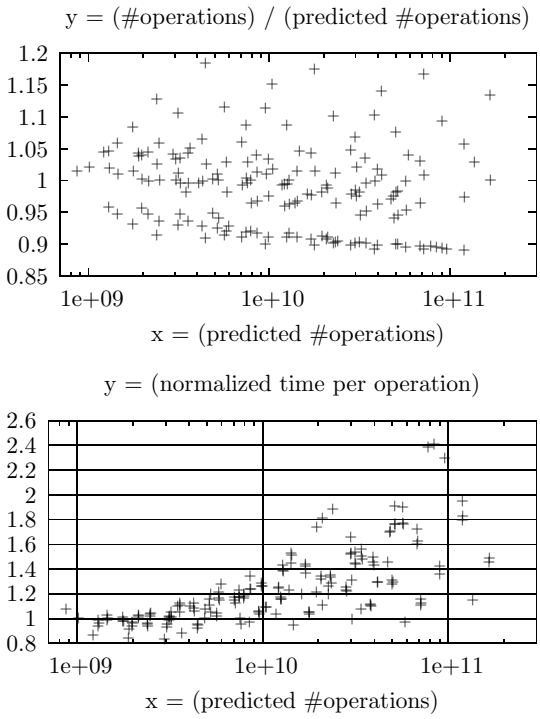
The plot below shows the run time in seconds, divided by the predicted time (the predicted number of basic operations (1) times the predicted time per basic operation):



The time exceeds the predicted time by up to a factor of two for large instances.

To understand this further, consider the next two plots. The plot on the left plots the actual the number of basic operations (obtained by instrumenting the code), divided by the estimate (1). The plot on the right plots the average time per operation.

[5] Preliminary experiments suggest that the more sophisticated CPLEX implementation is faster than GLPK Simplex, but, often, only by a factor of five or so. Also, preliminary experiments on larger instances than are considered here suggest that the running time of Simplex and interior-point methods, including CPLEX implementations on random instances grows more rapidly than estimated here.

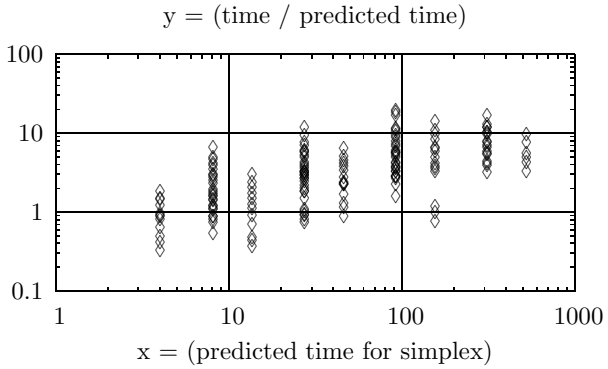


The conclusion seems to be that the number of basic operations is as predicted, but, unexpectedly, the *time per basic operation* is larger (by as much as a factor of two) for large inputs. We observed this effect on a number of different machines. We don't know why. Perhaps caching or memory allocation issues could be the culprit.

5.2 Empirical evaluation of Simplex

We estimate the time for Simplex to find a near-optimal approximation to be at least $5 \min(r, c)rc$ basic operations. This estimate comes from assuming that at least $\Omega(\min(r, c))$ pivot steps are required (because this many variables will be non-zero in the final solution), and each pivot step will take $\Omega(rc)$ time. (This holds even for sparse matrices due to rapid fill-in.) The leading constant 5 comes from experimental evaluation. This estimate seems conservative, and indeed GLPK Simplex often exceeded it.

Here's a plot of the actual time for Simplex to find a $(1 - \varepsilon)$-approximate solution (for each test input), divided by this estimate ($5 \min(r, c)rc$ times the estimated time per operation).



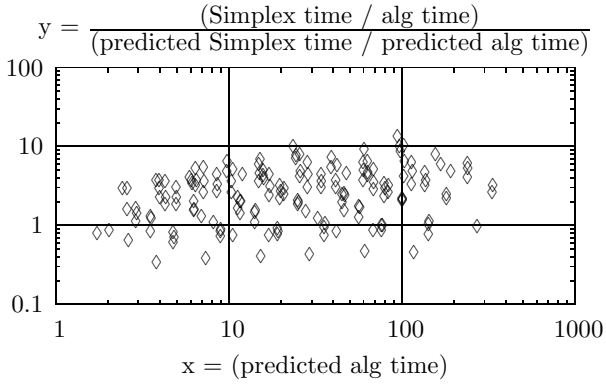
Simplex generally took at least the estimated time, and sometimes up to a factor of ten longer. (Note also that this experimental data excludes about 10% of the runs, in which GLPK Simplex failed to terminate due to basis cycling.)

5.3 Speed-up of this algorithm versus Simplex.

Combining the above estimates, a conservative estimate of the speed-up factor in using the algorithm here instead of Simplex (that is, the time for Simplex divided by the time for the algorithm here) is

$$\frac{5\min(r,c)rc}{[12(r+c) \ + \ 480d^{-1}]\ln(rc)/\varepsilon^2.} \tag{2}$$

The plot below plots the actual measured speed-up divided by the conservative estimate (2), as a function of the estimated running time of the algorithm here.



The speedup is typically at least as predicted in (2), and often more.

To make this more concrete, consider the case when $r \approx c$ and $\varepsilon = 0.01$. Then the estimate simplifies to about $(r/310)^2/\ln r$. For $r \geq 900$ or so, the algorithm here should be faster than Simplex, and for each factor-10 increase in r, the speedup should increase by a factor of almost 100.

5.4 Implementation issues

The primary implementation issue is implementing the random sampling efficiently and precisely. The data structures in [15, 10], have two practical drawbacks. The constant factors in the running times are moderately large, and they implicitly or explicitly require that the probabilities being sampled remain in a polynomially bounded range (in the algorithm here, this can be accomplished by rescaling the data structure periodically). However, the algorithm here uses these data structures in a restricted way. Using the underlying ideas, we built a data structure from scratch with very fast entry-update time and moderately fast sample time. We focused more on reducing the update time than the sampling time, because we expect more update operations than sampling operations. Full details are beyond the scope of this paper. An open-source implementation is at [19].

5.5 Data

The following table tabulates the details of the experimental results described earlier: "t-alg" is the time for the algorithm here in seconds; "t-sim" is the time for Simplex to find a $(1-\varepsilon)$-optimal soln; "t-sim%" is that time divided by the time for Simplex to complete; "alg/sim" is t-alg/t-sim.

r	c	k	100ε	t-alg	t-sim	t-sim%	alg/sim
739	739	2	2.0	1	3	0.31	0.519
739	739	2	1.0	7	6	0.51	1.251
739	739	2	0.5	33	7	0.64	4.387
739	739	5	2.0	3	1	0.51	2.656
739	739	5	1.0	15	1	0.63	8.840
739	739	5	0.5	63	2	0.76	30.733
739	739	4	2.0	2	2	0.51	0.970
739	739	4	1.0	11	3	0.64	3.317
739	739	4	0.5	46	4	0.76	11.634
739	739	3	2.0	2	3	0.43	0.561
739	739	3	1.0	9	5	0.60	1.745
739	739	3	0.5	38	6	0.72	6.197
1480	740	3	2.0	2	9	0.37	0.304
1480	740	3	1.0	13	13	0.53	0.959
1480	740	3	0.5	57	16	0.64	3.478
1480	740	2	2.0	2	24	0.44	0.102
1480	740	2	1.0	11	33	0.60	0.342
1480	740	2	0.5	51	39	0.71	1.313
1480	740	5	2.0	4	4	0.41	0.928
1480	740	5	1.0	18	6	0.56	2.930
1480	740	5	0.5	77	7	0.66	10.447
1480	740	4	2.0	3	6	0.34	0.495
1480	740	4	1.0	15	10	0.49	1.496
1480	740	4	0.5	64	12	0.60	5.239
740	1480	3	2.0	3	14	0.35	0.211
740	1480	3	1.0	14	21	0.51	0.667
740	1480	3	0.5	63	29	0.71	2.139
740	1480	2	2.0	2	13	0.27	0.192
740	1480	2	1.0	11	25	0.51	0.462
740	1480	2	0.5	54	34	0.68	1.597
740	1480	5	2.0	5	7	0.59	0.699
740	1480	5	1.0	22	9	0.72	2.460
740	1480	5	0.5	94	10	0.82	9.054
740	1480	1	2.0	2	23	0.24	0.097
740	1480	1	1.0	9	41	0.44	0.237
740	1480	1	0.5	47	55	0.59	0.848
740	1480	4	2.0	3	12	0.47	0.313
740	1480	4	1.0	17	15	0.61	1.130
740	1480	4	0.5	73	19	0.75	3.803

r	c	k	100ε	t-alg	t-sim	t-sim%	alg/sim
1110	1110	3	2.0	3	21	0.30	0.142
1110	1110	3	1.0	13	33	0.48	0.399
1110	1110	3	0.5	58	43	0.62	1.354
1110	1110	6	2.0	6	5	0.64	1.327
1110	1110	6	1.0	29	6	0.76	4.763
1110	1110	6	0.5	121	6	0.83	17.903
1110	1110	5	2.0	4	9	0.48	0.480
1110	1110	5	1.0	20	13	0.64	1.575
1110	1110	5	0.5	86	15	0.77	5.439
1110	1110	4	2.0	3	17	0.43	0.203
1110	1110	4	1.0	16	24	0.60	0.649
1110	1110	4	0.5	68	29	0.71	2.325
1111	2222	1	2.0	3	94	0.15	0.036
1111	2222	1	1.0	15	198	0.30	0.077
1111	2222	1	0.5	78	344	0.53	0.227
1111	2222	4	2.0	5	94	0.49	0.057
1111	2222	4	1.0	26	123	0.64	0.212
1111	2222	4	0.5	119	148	0.77	0.803
1111	2222	3	2.0	4	109	0.35	0.042
1111	2222	3	1.0	21	163	0.52	0.134
1111	2222	3	0.5	104	222	0.71	0.467
1111	2222	6	2.0	9	23	0.66	0.426
1111	2222	6	1.0	44	26	0.76	1.664
1111	2222	6	0.5	187	29	0.84	6.346
1111	2222	2	2.0	3	83	0.18	0.047
1111	2222	2	0.5	91	269	0.57	0.339
1111	2222	5	2.0	6	63	0.57	0.110
1111	2222	5	1.0	32	77	0.69	0.415
1111	2222	5	0.5	140	88	0.79	1.594
2222	1111	4	2.0	4	53	0.38	0.092
2222	1111	4	1.0	23	75	0.54	0.311
2222	1111	4	0.5	107	91	0.65	1.185
2222	1111	3	2.0	4	53	0.29	0.080
2222	1111	3	1.0	21	84	0.46	0.253
2222	1111	3	0.5	97	115	0.63	0.848
2222	1111	6	2.0	7	21	0.49	0.373
2222	1111	6	1.0	34	26	0.61	1.297
2222	1111	6	0.5	148	30	0.71	4.816
2222	1111	2	2.0	3	102	0.36	0.037
2222	1111	2	1.0	17	139	0.49	0.127
2222	1111	2	0.5	88	173	0.61	0.513
2222	1111	5	2.0	5	42	0.41	0.141
2222	1111	5	1.0	27	57	0.56	0.472
2222	1111	5	0.5	120	70	0.68	1.696

r	c	k	100ε	t-alg	t-sim	t-sim%	alg/sim
1666	1666	4	2.0	5	117	0.40	0.045
1666	1666	4	1.0	24	163	0.56	0.153
1666	1666	4	0.5	111	201	0.69	0.554
1666	1666	3	2.0	4	112	0.29	0.040
1666	1666	3	1.0	21	185	0.48	0.114
1666	1666	3	0.5	98	245	0.64	0.400
1666	1666	6	2.0	8	42	0.51	0.210
1666	1666	6	1.0	38	55	0.66	0.697
1666	1666	6	0.5	165	63	0.76	2.612
1666	1666	2	2.0	3	109	0.20	0.036
1666	1666	2	1.0	18	221	0.41	0.083
1666	1666	2	0.5	88	313	0.58	0.282
1666	1666	5	2.0	6	82	0.44	0.080
1666	1666	5	1.0	29	109	0.58	0.269
1666	1666	5	0.5	130	133	0.71	0.981
1666	3332	2	2.0	5	354	0.12	0.017
1666	3332	2	1.0	30	857	0.29	0.036
1666	3332	2	0.5	162	1594	0.54	0.102
1666	3332	5	2.0	9	509	0.51	0.020
1666	3332	5	1.0	51	654	0.65	0.078
1666	3332	5	0.5	227	762	0.76	0.299
1666	3332	1	2.0	5	350	0.09	0.015
1666	3332	1	1.0	24	1003	0.25	0.025
1666	3332	1	0.5	135	1881	0.46	0.072
1666	3332	4	2.0	7	578	0.38	0.014
1666	3332	4	1.0	42	899	0.58	0.047
1666	3332	4	0.5	204	1087	0.71	0.188
1666	3332	3	2.0	6	533	0.20	0.013
1666	3332	3	1.0	36	1095	0.41	0.033
1666	3332	3	0.5	180	1741	0.65	0.104
1666	3332	6	2.0	13	255	0.56	0.051
1666	3332	6	1.0	60	319	0.70	0.190
1666	3332	6	0.5	271	361	0.79	0.752
3332	1666	5	2.0	9	275	0.38	0.033
3332	1666	5	1.0	45	392	0.54	0.115
3332	1666	5	0.5	213	482	0.66	0.441
3332	1666	4	2.0	7	274	0.30	0.028
3332	1666	4	1.0	40	414	0.45	0.097
3332	1666	4	0.5	195	556	0.60	0.352
3332	1666	3	2.0	6	316	0.24	0.020
3332	1666	3	1.0	34	544	0.41	0.063
3332	1666	3	0.5	178	703	0.53	0.254
3332	1666	6	2.0	11	154	0.39	0.071
3332	1666	6	1.0	52	218	0.56	0.238
3332	1666	6	0.5	233	273	0.70	0.854

r	c	k	100ε	t-alg	t-sim	t-sim%	alg/sim
2499	2499	2	2.0	5	530	0.13	0.011
2499	2499	2	1.0	29	1556	0.40	0.019
2499	2499	2	0.5	159	2275	0.58	0.070
2499	2499	5	2.0	9	580	0.42	0.016
2499	2499	5	1.0	46	793	0.58	0.059
2499	2499	5	0.5	217	960	0.70	0.227
2499	2499	4	2.0	8	662	0.31	0.012
2499	2499	4	1.0	42	1064	0.50	0.040
2499	2499	4	0.5	195	1369	0.64	0.143
2499	2499	7	2.0	17	125	0.50	0.139
2499	2499	7	1.0	76	162	0.65	0.475
2499	2499	7	0.5	327	190	0.77	1.715
2499	2499	3	2.0	6	618	0.18	0.011
2499	2499	3	1.0	35	1079	0.32	0.032
2499	2499	3	0.5	174	1774	0.53	0.099
2500	5000	6	2.0	19	2525	0.52	0.008
2500	5000	6	1.0	98	3337	0.69	0.029
2500	5000	6	0.5	458	3828	0.79	0.120
2500	5000	7	2.0	26	1042	0.60	0.026
2500	5000	7	1.0	124	1272	0.73	0.098
2500	5000	7	0.5	556	1427	0.82	0.390
5000	2500	3	2.0	10	2165	0.23	0.005
5000	2500	3	1.0	62	3828	0.40	0.016
5000	2500	3	0.5	338	5586	0.58	0.061
5000	2500	6	2.0	17	1352	0.39	0.013
5000	2500	6	1.0	90	1832	0.53	0.049
5000	2500	6	0.5	418	2297	0.66	0.182
5000	2500	5	2.0	14	1752	0.33	0.008
5000	2500	5	1.0	82	2592	0.49	0.032
5000	2500	5	0.5	397	3330	0.63	0.119
5000	2500	4	2.0	12	1916	0.26	0.006
5000	2500	4	1.0	70	3177	0.44	0.022
5000	2500	4	0.5	367	4197	0.58	0.087
3750	3750	7	2.0	23	1828	0.50	0.013
3750	3750	7	1.0	111	2343	0.64	0.047
3750	3750	7	0.5	506	2712	0.74	0.187
3750	3750	6	2.0	18	3061	0.40	0.006
3750	3750	6	1.0	91	4263	0.55	0.022
3750	3750	6	0.5	432	5279	0.68	0.082

6 Future directions

Can one extend the coupling technique to *mixed* packing and covering problems? What about the special case of $\exists x \geq 0; Ax \approx b$ (important for computer tomography). What about covering with "box" constraints (upper bounds on individual variables)? Perhaps most importantly, what about general (not explicitly given) packing and covering, e.g. to maximum multicommodity flow (where P is the polytope whose vertices correspond to all $s_i \to t_i$ paths)? In all of these cases, correctness of a natural algorithm is easy to establish, but the running time is problematic. This seems to be because the coupling approach requires that fast primal *and* dual algorithms of a particular kind must both exist. Such algorithms are known for each of the above-mentioned problems, but the natural algorithm for each dual problems is slow.

The algorithm seems a natural candidate for solving *dynamic* problems, or sequences of closely related problems (e.g. each problem comes from the previous one by a small change in the constraint matrix). Adapting the algorithm to start with a given primal/dual pair seems straightforward and may be useful in practice.

Can one use coupling to improve *parallel and distributed* algorithms for packing and covering (e.g. [14, 21]), perhaps reducing the dependence on ε from $1/\varepsilon^4$ to $1/\varepsilon^3$? (In this case, instead of incrementing a *randomly* chosen variable in each of the primal and dual solutions, one would increment *all* primal and dual variables deterministically in each iteration: increment the primal vector x by $\alpha\hat{p}$ and the dual vector $\hat{x}$ by αp for the maximal α so that the correctness proof goes through. Can one bound the number of iterations, assuming the matrix is appropriately preprocessed?)

Acknowledgments

Thanks to two anonymous referees for helpful suggestions. The first author would like to thank the Greek State Scholarship Foundation (IKY). The second author's research was partially supported by NSF grants 0626912, 0729071, and 1117954.

References

1. Arora, S., Hazan, E., Kale, S.: The multiplicative weights update method: A meta-algorithm and applications. Theory of Computing **8**, 121–164 (2012)
2. Bienstock, D.: Potential Function Methods for Approximately Solving Linear Programming Problems: Theory and Practice. Kluwer Academic Publishers, Boston, MA (2002)
3. Bienstock, D., Iyengar, G.: Solving fractional packing problems in $O(1/\varepsilon)$ iterations. In: Proceedings of the Thirty First Annual ACM Symposium on Theory of Computing, pp. 146–155. Chicago, Illinois (2004)
4. Chudak, F.A., Eleuterio, V.: Improved approximation schemes for linear programming relaxations of combinatorial optimization problems. In: Proceedings of the eleventh IPCO Conference, Berlin, Germany. Springer (2005)
5. Clarkson, K.L., Hazan, E., Woodruff, D.P.: Sublinear optimization for machine learning. In: Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, pp. 449–457. IEEE Computer Society (2010)
6. Clarkson, K.L., Hazan, E., Woodruff, D.P.: Sublinear optimization for machine learning. Journal of the ACM **59**(5) (2012)
7. Garg, N., Koenemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. SIAM Journal on Computing **37**(2), 630–652 (2007)
8. Garg, N., Könemann, J.: Faster and simpler algorithms for multicommodity flow and other fractional packing problems. In: Thirty Ninth Annual Symposium on Foundations of Computer Science. IEEE, Miami Beach, Florida (1998)
9. Grigoriadis, M.D., Khachiyan, L.G.: A sublinear-time randomized approximation algorithm for matrix games. Operations Research Letters **18**(2), 53–58 (1995)
10. Hagerup, T., Mehlhorn, K., Munro, J.I.: Optimal algorithms for generating discrete random variables with changing distributions. Lecture Notes in Computer Science **700**, 253–264 (1993). Proceedings 20th International Conference on Automata, Languages and Programming
11. Klein, P., Young, N.E.: On the number of iterations for Dantzig-Wolfe optimization and packing-covering approximation algorithms. Lecture Notes in Computer Science **1610**, 320–327 (1999). URL `citeseer.nj.nec.com/440226.html`
12. Könemann, J.: Fast combinatorial algorithms for packing and covering problems. Master's thesis, Universität des Saarlandes (1998)
13. Koufogiannakis, C., Young, N.E.: Beating simplex for fractional packing and covering linear programs. In the forty-eighth IEEE symposium on Foundations of Computer Science pp. 494–504 (2007). DOI 10.1109/FOCS.2007.62
14. Luby, M., Nisan, N.: A parallel approximation algorithm for positive linear programming. In: Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, pp. 448–457. San Diego, California (1993)
15. Matias, Y., Vitter, J.S., Ni, W.: Dynamic Generation of Discrete Random Variates. Theory of Computing Systems **36**(4), 329–358 (2003)
16. Nesterov, Y.: Smooth minimization of non-smooth functions. Mathematical Programming **103**(1), 127–152 (2005)
17. Nesterov, Y.: Unconstrained convex minimization in relative scale. Mathematics of Operations Research **34**(1), 180–193 (2009)
18. Todd, M.J.: The many facets of linear programming. Mathematical Programming **91**(3), 417–436 (2002)
19. Young, N.: Fast ptas for packing and covering linear programs. `https://code.google.com/p/fastpc/` (2013)
20. Young, N.E.: K-medians, facility location, and the Chernoff-Wald bound. In: Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 86–95. San Francisco, California (2000)
21. Young, N.E.: Sequential and parallel algorithms for mixed packing and covering. In: Forty Second Annual Symposium on Foundations of Computer Science, pp. 538–546. IEEE, Las Vegas, NV (2002)

7 Appendix: Utility Lemmas

The first is a one-sided variant of Wald's equation:

Lemma 9 *[20, lemma 4.1] Let K be any finite number. Let $x_0, x_1, \ldots, x_T$ be a sequence of random variables, where T is a random stopping time with finite expectation.*

If $\mathrm{E}[x_t - x_{t-1} \mid x_{t-1}] \le \mu$ and (in every outcome) $x_t - x_{t-1} \le K$ for $t \le T$, then $\mathrm{E}[x_T - x_0] \le \mu\,\mathrm{E}[T]$.

The second is the Azuma-like inequality tailored for random stopping times.

Lemma 10 *Let $X = \sum_{t=1}^{T} x_t$ and $Y = \sum_{t=1}^{T} y_t$ be sums of non-negative random variables, where T is a random stopping time with finite expectation, and, for all t, $|x_t - y_t| \le 1$ and*

$$\mathrm{E}\big[\, x_t - y_t \mid \textstyle\sum_{s<t} x_s, \sum_{s<t} y_s \,\big] \;\le\; 0.$$

Let $\varepsilon \in [0,1]$ and $A \in \mathbb{R}$. Then

$$\Pr\big[\, (1-\varepsilon)X \;\ge\; Y + A \,\big] \;\le\; \exp(-\varepsilon A).$$

Proof Fix $\lambda > 0$. Consider the sequence $\pi_0, \pi_1, \ldots, \pi_T$ where $\pi_t = 0$ for $t > \lambda \mathrm{E}[T]$ and otherwise

$$\pi_t \doteq \prod_{s \le t} (1+\varepsilon)^{x_s}(1-\varepsilon)^{y_s} = \pi_{t-1}(1+\varepsilon)^{x_t}(1-\varepsilon)^{y_t} \le \pi_{t-1}(1+\varepsilon x_t - \varepsilon y_t)$$

(using $(1+\varepsilon)^x(1-\varepsilon)^y \le (1+\varepsilon x - \varepsilon y)$ when $|x-y| \le 1$).

From $\mathrm{E}[x_t - y_t \,|\, \pi_{t-1}] \le 0$, it follows that $\mathrm{E}[\pi_t \,|\, \pi_{t-1}] \le \pi_{t-1}$.

Note that, from the use of λ, $\sum_{s \le t} x_s - y_s$ and (therefore) $\pi_t - \pi_{t-1}$ are bounded. Thus Wald's (Lemma 9), implies $\mathrm{E}[\pi_T] \le \pi_0 = 1$.

Applying the Markov bound,

$$\Pr[\pi_T \ge \exp(\varepsilon A)] \le \exp(-\varepsilon A).$$

So assume $\pi_T < \exp(\varepsilon A)$. Taking logs, if $T \le \lambda \mathrm{E}[T]$,

$$X \ln(1+\varepsilon) - Y \ln(1/(1-\varepsilon)) = \ln \pi_T < \varepsilon A.$$

Dividing by $\ln(1/(1-\varepsilon))$ and applying the inequalities $\ln(1+\varepsilon)/\ln(1/(1-\varepsilon)) \ge 1-\varepsilon$ and $\varepsilon/\ln(1/(1-\varepsilon)) \le 1$, gives $(1-\varepsilon)X < Y + A$. Thus,

$$\Pr[(1-\varepsilon)X \ge Y + A] \le \Pr[T \ge \lambda \mathrm{E}[T]] + \Pr[\pi_T \ge \exp(\varepsilon A)] \le 1/\lambda + \exp(-\varepsilon A).$$

Since λ can be arbitrarily large, the lemma follows. □